به نام خدا

درس: برنامه سازی پیشرفته

ترم: ۹۹۲

استاد: دکتر مرتضی یوسفی صنعتی

دانشجو: محمد مرادی

ش دانشجویی: ۹۸۱۲۳۵۸۰۳۴

شرح تمرین

هدف : کار با رشته ها

std::array ، کلاس

و کلاس std::string

نام برنامه : StringTailor

یک کنسول اپلیکیشن که با

دستورات مشخص روی رشته

های ورودی پردازش انجام داده

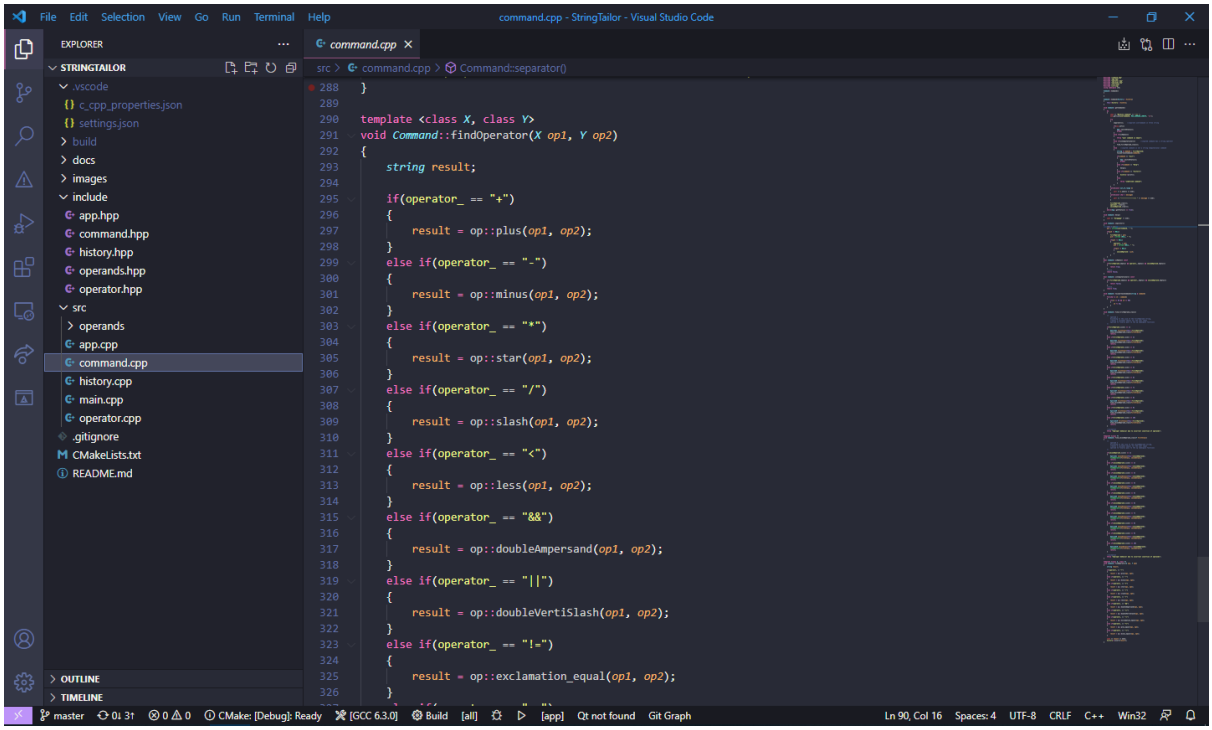
و نتیجه را چاپ می کند.

جزئیات ساخت برنامه

برنامه StringTailor در محیط ادیتور vscode درون سیستم عامل ویندوز کد نویسی و با کامپایلر GCC ۶.۳.۰ کامپایل شده است. همچنین برای اطمینان از خطاهای احتمالی، برنامه درون سیستم عامل اوبونتو هم با کامپایلر GCC ۹.۳.۰ آزمایش شد.

دستورات مجاز برنامه با دستور help نمایش داده میشوند.

در این پروژه از سیستم ساخت (build system) ، Cmake استفاده شده است.



روند اجرای برنامه

```
#include "app.hpp"

int main()
{
    App app;
    return app.exec();
}
```

با اجرای تابع exec از کلاس App برنامه آغاز شده و داده های اصلی برنامه پیکربندی میشوند.

سپس با فرخوانی تابع Command::getCommand() برنامه آماده دریافت دستور از کاربر میشود.

برنامه هر خط از دستورات کاربر را در یک آرایه کاراکتری ریخته و در نهایت وارد رشته های جداگانه می کند تا طول عملوند ها اعتبار سنجی شوند و در صورت لزوم پیغام خطایی چاپ شود. همچنین

باید برای فراخوانی توابع فضای نام op (توابع انجام عملیات روی عملوند های هر عملگر)، نوع عملگر عملیات مشخص گردد.

یادآوری: در متن تمرین خواسته شده است که برنامه برای نگهداری خط ورودی از آرایه استفاده کند اما نوع آرایه کاراکتری معمولی یا کلاس std::array مشخص نشده است لذا برای راحتی کار با آرایه های زبان c از آرایه کاراکتری استفاده شده است که تابع Command::separator هم به اقتضای آن از strtok() استفاده می کند.

کلاس Command

این کلاس وظیفه دریافت و پردازش دستور از کاربر را دارد.

```
#ifndef COMMAND_IG
#define COMMAND_IG

#include <string>
#include <array>
#include <history.hpp>
#define MAX_COMMAND_LENGTH 1000

class Command
{
    public:
        Command();
        Command(History *);
        void getCommand();
    private:
        char userCommand[MAX_COMMAND_LENGTH];
        std::string firstOperand;
        std::string operator_;
        std::string secondOperand;
        History * history;

        void help();
        void separator();
        bool isEmpty() const;
        bool isComputational() const;
        void toLowerCaseCommand(std::string &);

        void find_firstOperand_class();
        template <class T> void find_secondOperand_class(T);
        template <class X, class Y> void findOperator(X, Y);
};

#endif
```

آرایه کاراکتری userCommand توسط تابع getCommand از کاربر گرفته میشود و سپس توسط تابع separator نسبت به فاصله " " از هم جدا شده و در سه شیئ std::string که برای ذخیره سازی عملگر و عملوند ها هستند ریخته میشود.

تابع isComputational بررسی میکند که آیا دستور یه عبارت محاسباتی(دارای عملگر) است یا یک دستور از خود برنامه است.

تابع های find_firstOperand_class و find_secondOperand_class با توجه به تعداد کاراکتر های عملوند های اول و دوم، شیئ مناسبی از مجموعه کلاس های Operand می سازند تا به تابع findOperator ارسال شوند و در نهایت پس از تشخیص عملگر، تابع مناسب آن از فضای نام op فراخوانی شود.

```
do
{
    cout << "Machine Command" << ":=) ";
    cin.getline(userCommand, MAX_COMMAND_LENGTH, '\n');

    try
    {
        separator();    //separate userCommand in three string

        if(cin.eof())
        {
            App::switchStatus();
            break;
        }
        else if(isEmpty())
        {
            throw "your command is empty";
        }
        else if(isComputational())      //inputed command has a string operator
        {
            find_firstOperand_class();
        }
        else    //inputed command is not a string Computational command
        {
            string & command = firstOperand;
            toLowerCaseCommand(command);

            if(command == "exit")
            {
                App::switchStatus();
                break;
            }
```

تابع getCommand و دریافت و تشخیص نوع دستور

توابع گلوبال فضای نام op

برای هر عملیات برنامه تابعی عمومی موجود در این فضای نام نوشته شده است تا با فراخوانی پردازش ممکن را روی رشته ورودی انجام دهد.

```
#ifndef OPERATOR_IG
#define OPERATOR_IG

#include <string>

namespace op
{
    //one character
    template <class T, class C> std::string plus(T , C);
    template <class T, class C> std::string minus(T , C);
    template <class T, class C> std::string star(T , C);
    template <class T, class C> std::string slash(T , C);
    template <class T, class C> std::string less(T , C);
    //two character
    template <class T, class C> std::string doubleAmpersand(T , C);
    template <class T, class C> std::string doubleVertiSlash(T , C);
    template <class T, class C> std::string exclamation_equal(T , C);
    template <class T, class C> std::string plus_equal(T , C);
    template <class T, class C> std::string minus_equal(T , C);
};

#include "../src/operator.cpp"
/*  The template functions must be implemented in the header file,
    but has the different extension here .cpp Has been named and moved in src/ directory
*/
#endif
```

توابع انجام عملیات های هر عملگر به صورت template در نظر گرفته شده اند تا با توجه به طول ورودی، شیئ مناسبی از مجموعه ۱۰ کلاس Operand ساخته و به آنها ارسال شود.

کلاس های Operand

```cpp
#include <string>

/* Because the use of inheritance was not allowed
   10 classes with similar and repetitive definitions have been used
   and Their implementation is done in 10 cpp files in src/operands/
   directory */

class Operand1
{
    public:
        Operand1(std::string);
        std::string getText() const;
        size_t getSize() const;
    private:
        std::string text;     //user inputed operand
        void setText(std::string);
};
```

کلاس های Operand با توجه به مجاز نبودن استفاده از ارث بری در این تمرین پیاده و از ۱ تا ۱۰ شماره گذاری شده اند.

تاریخچه برنامه

```cpp
#ifndef HISTORY_IG
#define HISTORY_IG

#include <vector>
#include <string>

class History
{
    public:
        void insert(std::string);
        void print() const;
    private:
        std::vector <std::string> data;
};

#endif
```

برنامه دارای تاریخچه ای است که نتیجه های برنامه را در خود نگه میدارد و در صورت ورود دستور history چاپ میشوند.

کلاس History دارای وکتوری برای ذخیره این نتایج و همچنین توابعی عضو برای پردازش و عملیات روی این تاریخچه است.

تابع print از این کلاس تمامی تاریخچه نتایج محاسباتی برنامه را از آخر به ابتدا چاپ میکند.

ارسال

فایل های تمرین از طریق سامانه درس افزار ارسال شد.

cw\.basu.ac.ir

همچنین فایل ها درون مخزن گیت هاب به نشانی زیر موجود است.

https://github.com/itismoradi/StringTailor